Procedures and Guidelines

DIRECTIVE NO. 564-PG- 8700.2.1
EFFECTIVE DATE: 09/23/1998
EXPIRATION DATE: 09/23/2003

APPROVED BY Signature: original signed by
NAME: Robert Kasa
TITLE: Branch Head

Responsible Office: 564 / Microelectronics and Signal Processing Branch

Title: ASIC/FPGA Design Guide

1 PURPOSE

The purpose of this document is to describe procedure/steps to design Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs).

2 REFERENCES

- Guidelines for Writing VHDL Models in a team Environment, by Janick Bergeron, Bell-Northern research Ltd.
- VHDL Modeling Guidelines – European Space Agency
- VHDL Style Guide, by Troy R. Pesola
- VHDL Coding Styles and Methodologies – Ben Cohen, 1995, Kulwer Academic Publishers
- Synopsys Document: METH v1.0a: Current Methodologies Note: HDL
- Synopsys Pitfalls and How to Avoid Them
- Java Style Guide – July 1997 – DSTL-97-002 Published by NASA/GSFC
- C Style Guide – August 1994 Software Engineering Laboratory Series SEL-94-003
- The NASA ASIC Guide: Assuring ASICs for Space

3 SCOPE

The scope of the procedure described in this document is to guide ASIC/FPGA designers to follow a streamlined process to design an ASIC or a FPGA from the concept level to a final product.

4 IMPLEMENTATION

The use of Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs) in common ground and space electronics is rapidly increasing. The project managers/leaders must understand and prepare to deal with wide range of complex issues.

In today' s electronic systems, ASICs and FPGAs offer many different advantages over off -the-shelf devices based on the type of applications each device is being used for. The advantages of ASICs include:

- higher performance
- reduced board space
- lower costs
- reduced power and heat

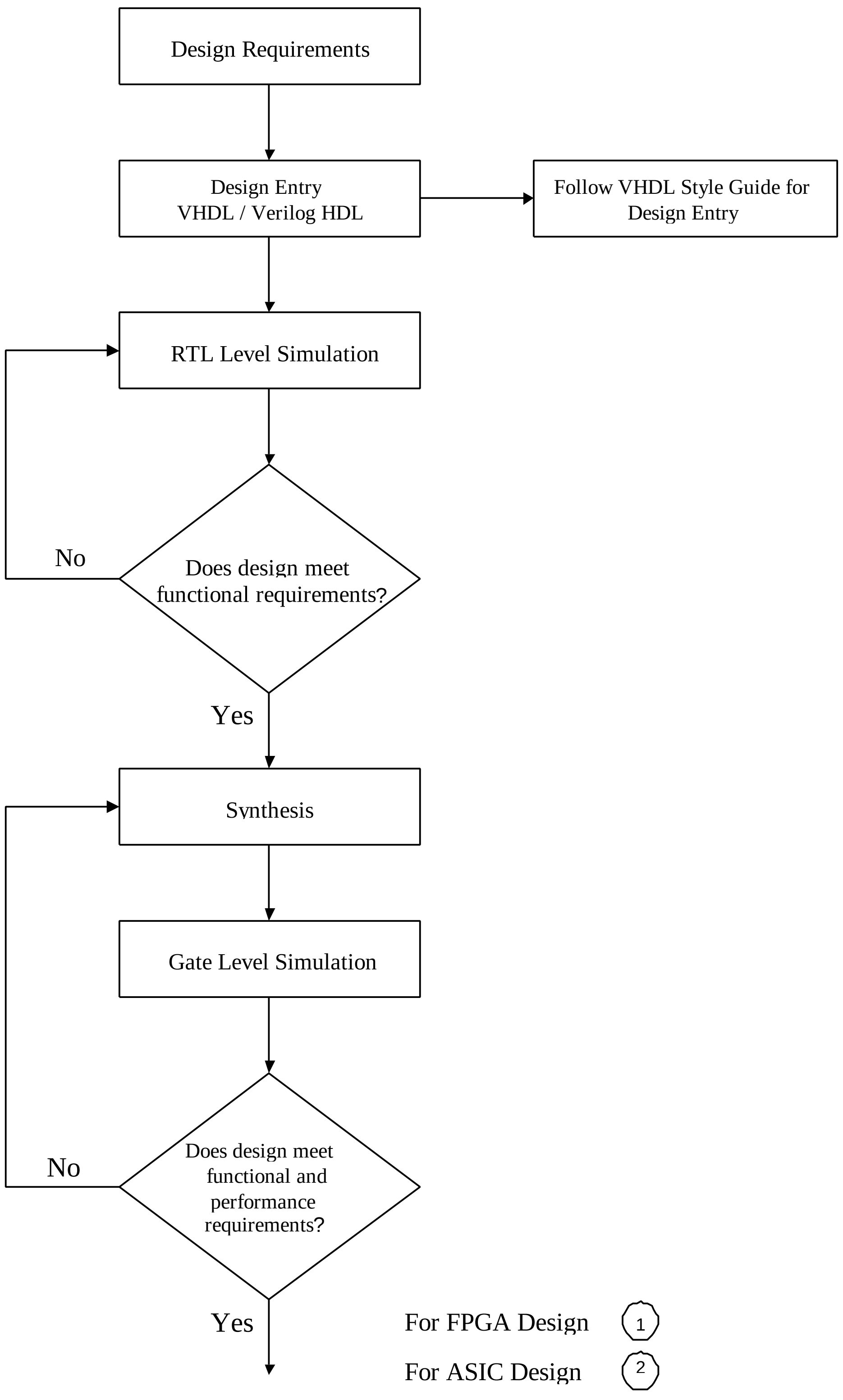
- more testability features
- higher reliability

The FPGAs offer many advantages over ASICs and off-the-shelf devices such as
- reduced design cycle time
- lower costs
- reprogramming capabilities
- lower power

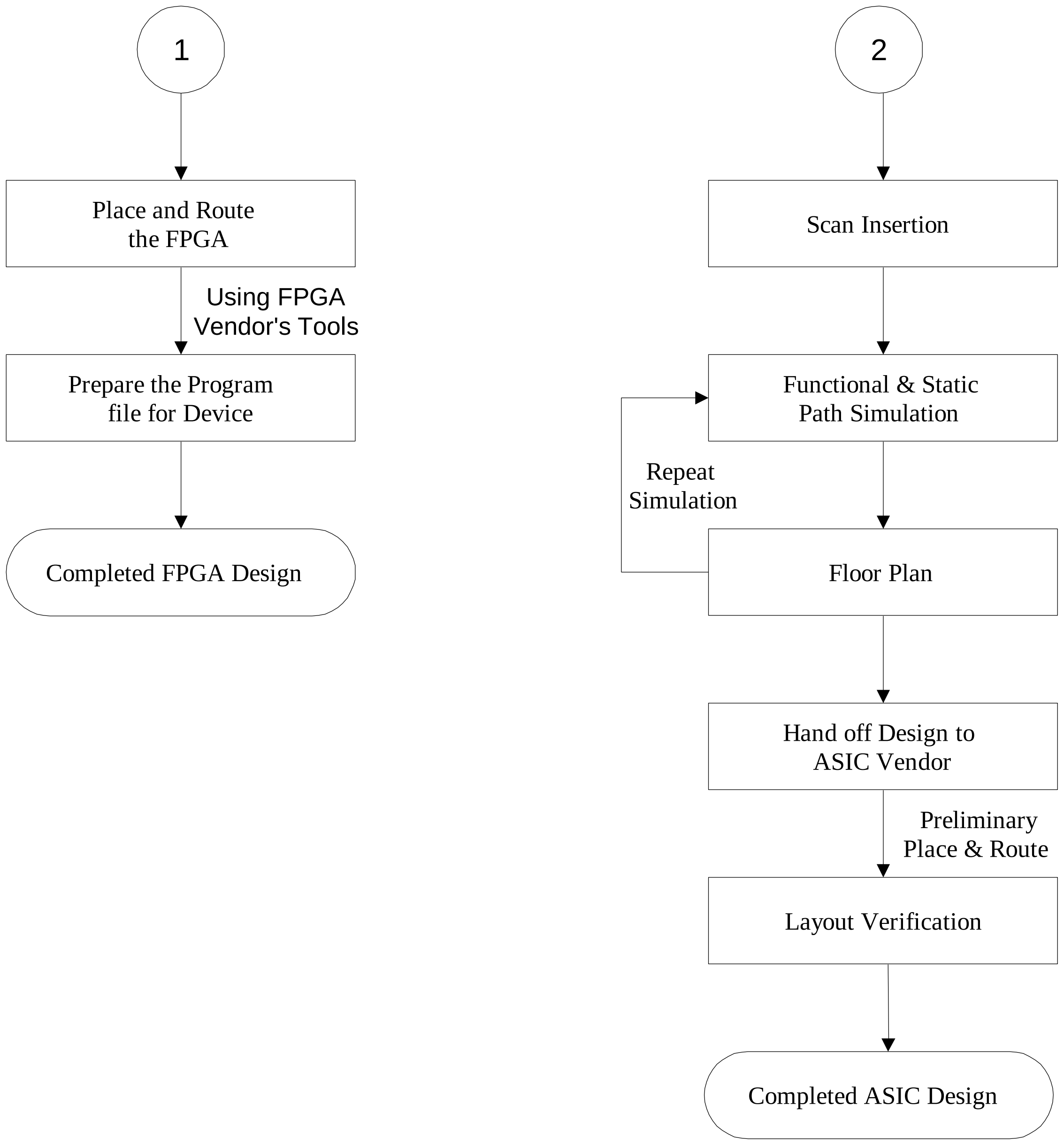
The success of ASIC/FPGA design rests upon up-front planning and regular guidance. The meaning of success is delivering a usable part without multiple passes of silicon, budget overruns and schedule slips. It is necessary to understand the entire design process i.e. from requirements to testing of the silicon, in order to make a successful ASIC design.

The ASIC/FPGA design flow chart (shown on page 2) illustrates the steps and sequence essential for a successful ASIC/FPGA design.

ASIC/FPGA Design Flowchart


Design Requirements
Design Entry VHDL / Verilog HDL
Follow VHDL Style Guide for Design Entry
RTL Level Simulation
Does design meet functional requirements?
No
Yes
Synthesis
Gate Level Simulation
Does design meet functional and performance requirements?
No
Yes
For FPGA Design
1
For ASIC Design
2

VHDL Design Flowchart


1
Place and Route the FPGA
Using FPGA Vendor's Tools
Prepare the Program file for Device
Completed FPGA Design
2
Scan Insertion
Functional & Static Path Simulation
Repeat Simulation
Floor Plan
Hand off Design to ASIC Vendor
Preliminary Place & Route
Layout Verification
Completed ASIC Design

VHDL Style Guide

Version 1.0

Introduction

This document represents a collection of style guidelines extracted from the documents listed in the references section as well as other guidelines developed within Code 564. The reading of the referenced documents is highly recommended. It is expected that this guide will change as these guidelines are applied to our design environment. The following are the objectives of this VHDL style guide:

- Create guidelines which support the development of applications in a team environment. This includes:
  1. A hierarchical design approach
  2. Incremental design, integration & testing of applications
  3. Synthesizable code creation
- Create a common style to facilitate code readability and re-usability.
- Facilitate the development of code which can be:
  - ◊ re-targeted to different architectures as required.
  - ◊ synthesized with various tools across different operating systems.

Project Organization

- A file should contain only one design unit. A design unit is defined as any of the following:
  - ◊ entity
  - ◊ architecture
  - ◊ entity - architecture pair
  - ◊ package
  - ◊ package body
  - ◊ test bench
  - ◊ configuration (may be combined with one of the above design units if it improves the readability of the design).
- The structure of the file system hierarchy shall mirror the logical structure of the system being modeled.
- A directory shall correspond to one and only one library.
- A directory shall have the same name as the topmost design unit it contains.
- If the VHDL system requires a directory or a file for its own data or object files, it shall be named toolName.lib
- A complete copy of the entire hierarchy of the model shall be kept in a shared central location.
- Each member of the development team shall have a copy of only the portion of the hierarchy for which (s)he is responsible for. Each member creates a complete picture of the model by having soft links to the missing portions in the central repository.
- Each directory shall contain a make file. A rule name all should compile **all** units contained in the directory and invoke make for all sub directories. If several toolsets are used, the make files

should be named Makefile.toolName and a soft link named Makefile should point to the make file for the tool currently used.

File Naming Conventions

- All files that contain VHDL source should have a .vhd extension.
- All scripts should have a .scr extension.
- Each file should be named according to the design unit(s) it contains:

◊ entity/architecture	=> basename.vhd
◊ configuration	=> basename_cfg.vhd
◊ package	=> nametypes.vhd where name is the FPGA/ASIC name
◊ test bench	=> testbasename.vhd

- The base name should be the same as the design unit name the file contains.
- The name of a VHDL library shall be directoryName_LIB.

File Prolog

The following prolog should be included at the top of every VHDL source file.

```
----------------------------------------------------------------
-- File name
----------------------------------------------------------------
-- Directory path were file can be found
--
-- This VHDL code was developed by NASA,
-- Goddard Space Flight Center, Code 564
-- for the Project Name project.
--
----------------------------------------------------------------
-- Description
----------------------------------------------------------------
-- Explanation of the functionality of the code
--
----------------------------------------------------------------
-- Notes
----------------------------------------------------------------
-- Anything relevant about this code, including document
-- references, assumptions, constraints, restrictions, etc.
--
----------------------------------------------------------------
-- Development History
----------------------------------------------------------------
-- (most recent date first)
-- Date Author Reference
-- Description
--
-- Date Author Reference
-- Description
--
----------------------------------------------------------------
-- Dependencies
----------------------------------------------------------------
-- File names of VHDL entities referred in this code
-- File names of associated script and data files
-- Special simulation and synthesis commands
--
----------------------------------------------------------------
-- Warning
----------------------------------------------------------------
-- This VHDL description is property of the National
-- Aeronautics and Space Administration. Unauthorized use or
-- duplication of this VHDL description is strictly
-- prohibited. Authorized users are subject to the following
-- restrictions:
--
--      - Neither the author, their corporation, nor NASA is
--        responsible for any consequence of the use of this
--        VHDL description.
--
--      - The origin of this VHDL description must not be
--        misrepresented either by explicit claim or by omission.
--
--      - Altered versions of this VHDL description must be
--        plainly marked as such.
--
--      - This notice may not be removed or altered.
--
----------------------------------------------------------------
----------------------------------------------------------------
```

Code Organization

Indentation

- Indentation should be used to show logical structure of code.
- 3 spaces should be used for each level of indentation.
- Do not use tab characters. Instead configure your editor to expand tabs into spaces.
- Code should start in the left-most column and be indented for each block.
- Indentation levels in sequential statements shall not exceed 4. If more levels are required, the design should be separated into more manageable parts.
- Indented regions in sequential statements shall not have more than 60 lines.
- When including libraries and packages, indent the use statement from the library declaration.
- Labels should be placed in the left-most column.

Comments

- Comments are used to provide information that a person could not discern simply by reading the code. The purpose of comments is to allow the function of a design to be understood by a designer not involved in the development of the code.
- Comments shall be on a line of their own unless they fit within the 80 character maximum line length.
- Multi-line comments shall start and end with an empty comment line. Empty comment lines extended to the right of the page can be used to clearly separate sections of code.
- Comments shall be immediately followed by the code they describe

Managing File Size

- Use a hierarchical design approach to keep files to a manageable size.
- Each file should contain no more than one entity-architecture pair.
- Component declarations should appear in packages only.
- Lines shall not exceed 80 characters in length. If more than 80 characters are required continue the statement on the next line.
- Long lines shall be broken where there is white spaces.
- Line continuations shall be indented to line-up with the first token at the same nesting level or by 3 spaces.
- There shall be no more than 2 process statements in an architecture. Use a structural decomposition if necessary.

VHDL Coding Style

White Space

- Add white space in the form of blank lines, spaces, and indentation to improve the readability of the code.
- Concurrent statements (i.e. separate processes) and their descriptive comments shall be separated by 2 blank lines.
- Groups of logically related statements and declarations shall be separated by 1 blank line.
- Choices in a case statement shall be separated by 1 blank line.
- Unless otherwise specified, tokens shall be separated by 1 space. No space shall precede a close parenthesis, comma, colon, or semi-colon nor follow an open parenthesis and no space shall surround a single quote or dot.
- Each statement shall start on a new line.

Alignment

- Elements in interface declarations shall be vertically aligned.
- Elements in named associations that span more than one line shall be vertically aligned.
- Assignments operators within the same block should line up vertically.

Labels

- Concurrent statements shall be labeled.
- Loop statements shall be labeled.
- Next and exit statements shall specify the loop label they control.
- Whenever possible END keywords shall be qualified
- Named association shall be used preferably to position association.

Component Instantiation

- All components shall be explicitly instantiated.
- A component instance name should be composed of the component name and the component main output signal name separated by an underscore. For example, a multiplier with a signal named Q as the product would be called MULT_Q.

Object Naming Conventions

- Objects should be named using the format *name_polarity_class* in all upper case letters. The *name* should be a meaningful, descriptive term for the object. Use underscores to separate multiple words within the name. *Polarity* may be omitted if the signal is active high. For active low signals, use _NEG for the *polarity*. If the class of the object is omitted, it is assumed to be a signal (_S). The following suffixes should be used for the _*class* of the object:
    - _S => signal
    - _C => constant
    - _V => variable
    - _IDX => loop index
    - _I => internal signal
    - _F => file
    - _G => generic
    - _LIB => library
    - _LBL => label
    - _A => architecture

      The following may be used in place of _A for architectures:
    - _BEH => behavioral architecture
    - _RTL => register-transfer level (synthesizable) architecture
    - _STR => structural architecture
    - _CFG for configuration
    - _TYP for user-defined type and subtype identifiers
    - _PKG for user-defined package identifiers
- Use lower case letters for all VHDL reserved words, attributes, etc.
- VHDL component names should be composed by the three initial letters of the function of the component followed by information essential to the identification of the component.
    - ADD16S - 16-bit signed adder

  - ♦ ADD16RU - 16-bit unsigned adder, registered output
  - ♦ CMP16U - 16-bit unsigned comparator
  - ♦ CMP16S - 16-bit signed comparator
  - ♦ CNTUP16 - 16-bit up counter
  - ♦ CNTDN16 - 16-bit down counter
  - ♦ DIV8X8U - 8x8 unsigned divider
  - ♦ DIV8X8S - 8x8 signed divider
  - ♦ DLY5_16 - 16-bit 5 clocks delay element
  - ♦ MUL8x8U - 8x8 unsigned multiplier
  - ♦ MUL8X8S - 8x8 signed multiplier
  - ♦ MUX18x3 - 18-bit 3-inputs multiplexer
  - ♦ SUB16S - 16-bit signed subtractor
  - ♦ SUB16RU - 16-bit unsigned subtractor, registered output
  - ♦ FUNCTION_CTRL controller
- The identifiers in predefined packages shall be used in uppercase.

Literals

- Only literals in base 2, 8, 10 or 16 shall be used.
- Underscore may be used in literals to improve readability.
- Extended digits in base 16 literals and base specifiers shall be in uppercase.
- Real literals shall be in decimal only.

Miscellaneous

- Group functions logically. For example, separate the data path and control sections into different entities.
- Buffer ports shall not be used.
- Block statements shall not be used.
- Operators shall not be overloaded lightly. Be extra careful not to mislead the reader of the code in respect to the operation performed.
- Attributes 'range and 'reverse_range shall be used when scanning arrays.
- Variable-width ports shall be constrained using generics.
- Enumerates shall be used to represent non-arithmetic discrete values.
- Constants shall be used to represent limits and parameters.
- The 'event attribute shall be used explicitly when testing for a change to a particular level. Avoid the use of rising_edge() and falling_edge() which are unsupported by some synthesis tools.

VHDL Recommendations

Simulation

- Use enumerated types when defining state variables. This will cause the state names to be displayed in the wave form viewer instead of the binary state value which makes the wave forms much easier to interpret. If you need to encode your states in a specific way (i.e. gray code, one-hot, etc.) specify this in your simulation script.

Synthesis

- Avoid asynchronous design practices. A synchronous design is one where all flip-flops are fed by a single clock that is a primary input to the design. Synchronous designs perform the same function regardless of the clock rate, as long as the rate is slow enough to allow signals to completely propagate through the combinational logic between flip-flops. While fully synchronous circuits generally agree with their simulation models, asynchronous circuits may not.
- Avoid the use of booleans and variables. Instead use STD_LOGIC and STD_LOGIC_VECTOR's.
- Use the case construct instead of nested if statements
- Sensitivity lists are a mechanism for improving simulation performance. Use them with care to avoid simulation/synthesis mismatches.
- Control circuit initialization with reset signals, otherwise simulation and synthesis initialization assumption differences cause discrepancies in the RTL and gate-level simulation results.
- When driving a synthesis tool, you should set the design goals (timing/performance) and not try to model the final implementation. (no **wait until** statements).
- Adopt an implication (directly specifying the logic structure) rather than an inference (functional specification of the solution) approach.
- To use an output signal on the right side of an assignment, create an internal signal to feed it back.

CHANGE HISTORY LOG

Revision	Effective Date	Description of Changes
Baseline	09/23/1998	Initial Release